An Intelligent Front End for Ecological Modelling

AN INTELLIGENT FRONT END FOR ECOLOGICAL MODELLING

Michael Uschold
Nigel Harding
Robert Muetzelfeldt
Alan Bundy

D A I Research Paper No. 223

Submitted to *ECAI-84*

AN INTELLIGENT FRONT END FOR ECOLOGICAL MODELLING

Michael Uschold, Robert Muetzelfeldt, Alan Bundy, Nigel Harding and David Robertson

Department of Artificial Intelligence and
Department of Forestry and Natural Resources
University of Edinburgh
Edinburgh, Scotland

Introduction

An intelligent front end is a user-friendly interface to a software package which would otherwise be technically incomprehensible and/or too complex to be accessible to many potential users. It allows the users to explain their problems in language familiar to them and then translates this into a language suitable for the software package. This paper describes ECO, an intelligent front end for ecological modelling.

The Ecological Modelling Context

An ecologist can use a computer model to predict the behaviour of an ecosystem and the effect of alterations within that ecosystem, e.g. s/he might investigate the movement of DDT through a food web and the effects of that movement on the organisms involved. Such models may: increase the precision of an ecologist's understanding of the environment; provide a means to test the validity of this understanding; assist management of natural resource systems; and assist the assessment of environmental impacts. More widespread use of ecological modelling is currently hampered by a number of factors. Realistic models are large and complex, requiring considerable mathematical and programming skills which many ecologists do not possess. Building on the work of others is difficult because existing models tend to be 'one-off' and the results are scattered in the literature. Hence, an intelligent front end for ecological modelling would be a valuable resource in ecology and for society in general.

Ecological modelling presents a challenging problem for knowledge engineering. Most existing intelligent front ends, such as SACON (Bennet and Englemore, 1979) and ELAS (Weiss et al,1982), use a production rule system to infer the appropriate instructions for a package from the user's description of the problem. This solution implies a small enough conceptual distance between the user's description of the problem and the instructions to the package, for them both to be expressible in the same language. However, in ECO the user cannot be assumed to be familiar with the mathematical or programming language necessary to describe the model. The major problem is the translation of the user's description into the model. Production rules do not seem suitable for this, although they may have a limited role to play in the reformulation of the user's description into a form suitable for translation.

System Dynamics Models

In the prototype ECO system we restricted our attention to those ecological models which can be represented in the notation of System Dynamics models (Forrester,1961). This methodology encompasses the technique of compartment modelling, commonly used in ecology to model the flow of materials such as energy, nutrients, and pollutants. System Dynamics modelling makes use of a concise schematic representation, which helps the ecologist think about the model without mathematical formulae, and it serves as a suitable starting point for a task specification formalism for ECO.

The Task Specification Formalism

The task specification is an intermediate representation which bridges the terminological gap between the user and the package, (in this case, the FORTRAN program). It must be fairly close to the user's view of the problem, while at the same time being sufficiently well-defined to permit translation into a program which produces mathematically correct simulation results for the desired system. The task specification encodes the user's ecological description as a well-defined mathematical model. Being near the user's view of the problem, it allows for a more natural dialogue to take place and thus minimizes the amount of processing necessary while creating and modifying it. Small changes in the user's problem description cause small changes in the task specification, even though they might imply considerable changes in the simulation program.

The Ecological Knowledge Base

The base of ecological data and relationships provides the user with the building blocks for making models and with guidance on how to put them together appropriately. The user consults this knowledge base directly when selecting modules. It is used to infer the "obvious", easing the burden on the ecologist of specifying every last detail. A third important use of the knowledge base is maintaining consistency in the model. For instance, the units of the quantities in the model must be compatible and conversions made when necessary. Modules selected must be applicable to the ecological context in which they are chosen.

Included in the knowledge base are:

- A module library, which contains a specification for various ecological relationships that the modeller might wish to include in his model;
- Information on ecological entities that the modeller might wish to include, such as sheep trees, etc.
- A process library, which defines the ecological processes that the user can refer to (such as photosynthesis, grazing), plus the ecological contexts in which they can be used.

The Dialogue Subsystem

The job of the dialogue handler is to create the task specification. A very flexible form of dialogue is required, since the space of all possible ecological problems is large and ill-structured. An EMycin-style dialogue system, in which the user reacts to production rule driven questions, is not suitable for describing an ecological environment.

The model-building process begins with broad statements which indicate the main features of the model. The mathematical structure is specified, and finally, all initial conditions and parameter values are filled in. The user has access to the ecological knowledge base which is used to assist in selecting appropriate modules and parameter values. To some extent, we relieve the user of having to specify every last detail. Whenever possible, the program deduces the "obvious" quietly, behind the scenes. For example, modules selected in certain situations have some of the inputs automatically filled in on the basis of what has been previously specified in the model.

The main types of statement that the user can make are:

- Process statements: Specifies a process taking place between one or more entities. Example: **sheep graze grass**
- USES: Specifies how a quantity is to be computed. Example: **photosynthesis uses light_response_equation**
- SET: Assigns a value to a quantity. Example: **set temperature=12**
- UNIFY: Two quantities in the model which had previously been assumed to be different are unified. Example: **unify temperature,air_temperature**
- DISPLAY: Displays the current state of the model, or a specified part of it.

Program Generation

The task of generating correct FORTRAN code to implement the model is greatly simplified because there is a reasonably well-defined program template which can be used to run System Dynamics models simulations. The following is such a template:

```
Read parameter values and initial contents of Compartments
Main Loop:  For each time increment:
   - Calculate flows
   - Update contents of compartments
Examine simulation results
```

This simple template will not be appropriate for ecological models beyond the framework of System Dynamics. The hard part is in calculating the flows. Each module is already precoded as a SUBROUTINE. Its arguments are the inputs to the module, followed by the outputs. The program generator must be able to output the correct sequence of subroutine calls so that variables used as inputs always have values previously assigned. It also has to get all the input and output variables in the appropriate argument slots.

Current Major Limitations/Future Work

The Process and Entities section in the knowledge base are toyfully small. Scaling these up is straightforward.

- We are developing an intelligent browser to guide the user through the large database of mathematical relationships and parameter values.
- We will provide help in selecting modules. For example, a statement such as "temperature depends on altitude" will cause ECO to suggest the module(s) or chain(s) of modules for calculating this relationship.
- We need to go beyond System Dynamics models. A more general, lower level modelling approach will have to be adopted: an appropriate corresponding task specification formalism is already being developed. It is imperative that the user interface remains at a high level, relatively close to the user's view of the model. Otherwise, the ecologist will be unable to create models conveniently if at all. Serious limitations of System Dynamics models include:
    * No facility for handling substructure exists. For instance, a deer population may be divided into several herds, each of which may be subdivided into age and/or sex classes.
    * No facility exists for doing calculations on varied time scales in the same model. Certain things ought to be calculated daily, others monthly, or yearly.

Acknowledgements

This research was supported by SERC grant, number GR/C/06226.

References

Bennet, J S. and Englemore, R. SACON: A Knowledge-Based Consultant for Structural Analysis. Proceedings of the sixth IJCAI, International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979, pp. 47-49.

Forrester, J.W. Industrial Dymanics. MIT Press, 1961.

Weiss S., Kulikowski C., Apte C., Uschold M., Patchett J., Brigham R., and Spitzer B. Building Expert Systems for Controlling Complex Programs. Proceedings of AAAI-82, American Association for Artificial Intelligence, 1982, pp. 322-326.

An Intelligent Front End for Ecological Modelling

by
Michael Uschold, Nigel Harding, Robert Muetzelfeldt and Alan Bundy

Acknowledgements

This research was supported by SERC grant, number GR/C/06226.

Abstract

In this paper, we describe ECO, an intelligent front end for ecological modelling. ECO helps an ecologist user build a customised, FORTRAN program which simulates the processes in an ecosystem.

The user interacts with ECO in a free-form dialogue about the environment using only ecological terminology. Together they put together a System Dynamics model using the user's input and ECO's on-line knowledge base of ecological modelling information. ECO translates this model into a FORTRAN program and runs it.

Classic production rule architectures were found inadequate for ECO, and we are developing alternative techniques.

Keywords

Expert System, Intelligent Front End, Dialogue, Program Generation, Ecological Modelling, Compartment Modelling, System Dynamics models.

1. Introduction

An intelligent front end is a user-friendly interface to a software package which would otherwise be technically incomprehensible and/or too complex to be accessible to many potential users. It allows the users to explain their problems in language familiar to them and then translates this into a language suitable for the software package. The classic intelligent front end is SACON, [Bennet and Englemore 79]: which advises on the use of a finite element package.

In this paper, we describe ECO, an intelligent front end for ecological modelling.

1.1. Motivation

An ecologist can use a computer model to predict the behaviour of an ecosystem and the effect of alterations within that ecosystem, e.g. s/he might investigate the movement of DDT through a food web and the effects of that movement on the organisms involved. Such models may: increase the precision of an ecologist's understanding of the environment; provide a means to test the validity of this understanding; assist management of natural resource systems; and assist the assessment of environmental impacts.

Some ecologicial models can be represented as sets of differential or difference

equations. These can be solved numerically by existing packages like CSMP, Dynamo or ACSL. However, many ecological models do not fit into these frameworks and must be represented with a customised, simulation program. So rather than build an interface to an inadequate package, ECO produces a simulation program in FORTRAN. Thus, ECO is not an intelligent front end in the strict sense (unless one counts the FORTRAN compiler as a package), but it seems sensible to label it as one since many of the issues we face would be the same if ECO did interface to a standard package.

More widespread use of ecological modelling is currently hampered by a number of factors. Realistic models are large and complex, requiring considerable mathematical and programming skills which many ecologists do not possess. Building on the work of others is difficult because existing models tend to be 'one-off' and the results are scattered in the literature. Hence, an intelligent front end for ecological modelling would be a valuable resource in ecology and for society in general.

Ecological modelling also presents a challenging problem for knowledge engineering. Most existing intelligent front ends (e.g. SACON, ELAS [Weiss et al 82]) use a production rule system to infer the appropriate instructions for a package from the user's description of the problem. This solution implies a small enough conceptual distance between the user's description of the problem and the instructions to the package, for them both to be expressible in the same language. However, in ECO the user cannot be assumed to be familiar with the mathematical or programming language necessary to describe the model. The major problem is the *translation* of the user's description into the model. Production rules do not seem suitable for this, although they may have a limited role to play in the reformulation of the user's description into a form suitable for translation.

A very flexible form of dialogue is required, since the space of all possible ecological problems is large and ill-structured. An EMycin-style dialogue system, in which the user reacts to production rule driven questions, is not suitable for describing an ecological environment. A more free-form dialogue is required, in which the user inputs particular relationships, e.g. sheep graze on grass, albeit in a formal language.

1.2. Design Considerations

These considerations demand a different system architecture from the standard expert system shell. In particular, they demand:

- an **intermediate representation** which encodes the user's, ecological description of the problem as a mathematical model (this is described in section 4);

- a **program generation** subsystem which translates the intermediate representation into FORTRAN code (this is described in section 7);

- a **knowledge base** of ecological modelling **modules** and other ecological data from which the mathematical model can be constructed (this is described in section 5);

- a free-form **dialogue handling** subsystem which enables users to describe the problem in their own terms, and to peruse the knowledge base to choose modules on the basis of their ecological properties (this is described in section 6); and

- an **analysis** subsystem which enables the results generated by running the FORTRAN program to be displayed and analysed by the user (this has not yet been built).

Some of these features were present in the MECHO system, [Bundy et al 79], for solving mechanics problems stated in English, namely the intermediate representation and a non-interactive synthesis subsystem. The two domains are sufficiently different that it was not possible to adapt the MECHO program, but MECHO has been run on a few ecological problems, and it has inspired some of the features of ECO.

The Φnix system, [Barstow et al 82], is tackling a similar kind of problem to ECO, but in the domain of oil log interpretation, and it seems to be adopting similar solutions. The users of Φnix have a wide space of possible problems and little or no knowledge of the modelling language. A graphics interface is used to allow the user to input the problem, and use program generation to build a FORTRAN simulation program. We expect many intelligent front end developers to encounter similar problems and hope that together we can build some general tools for such tasks by generalizing systems like ECO, MECHO and Φnix.

In the prototype ECO system we restricted our attention to those ecological models which can be represented in the notation of **System Dynamics** models [Forrester 61]. This methodology encompasses the technique of compartment modelling, commonly used in ecology to model the flow of materials such as energy, nutrients, and pollutants. System Dynamics modelling makes use of a concise schematic representation, which helps the ecologist think about the model without mathematical formulae, and it serves as a suitable starting point for the intermediate representation needed by ECO.

2. Structural Overview of ECO

The user begins by conversing with the dialogue subsystem. The knowledge base is at the core of this stage (Fig. 2-1). It contains the building blocks for the model, and information on how and when to use them. During this dialogue, the program builds the intermediate representation for the model, which is passed to the program generator, which in turn produces the FORTRAN code. The code is compiled, and the program is run, producing the results of the simulation. The user examines these results and may return to the dialogue subsystem to refine the model if desired.

3. System Dynamics Models

A System Dynamics model represents a system as a set of compartments with material flowing into and out of them. One can think of each compartment as a *tank*. Each *tank* has some filling pipes, and some emptying pipes which connect to other tanks. Each connecting pipe has a valve which governs the flow. Every System Dynamics model has one special "tank" called the source/sink which is the "outside world" with respect to the system being considered.

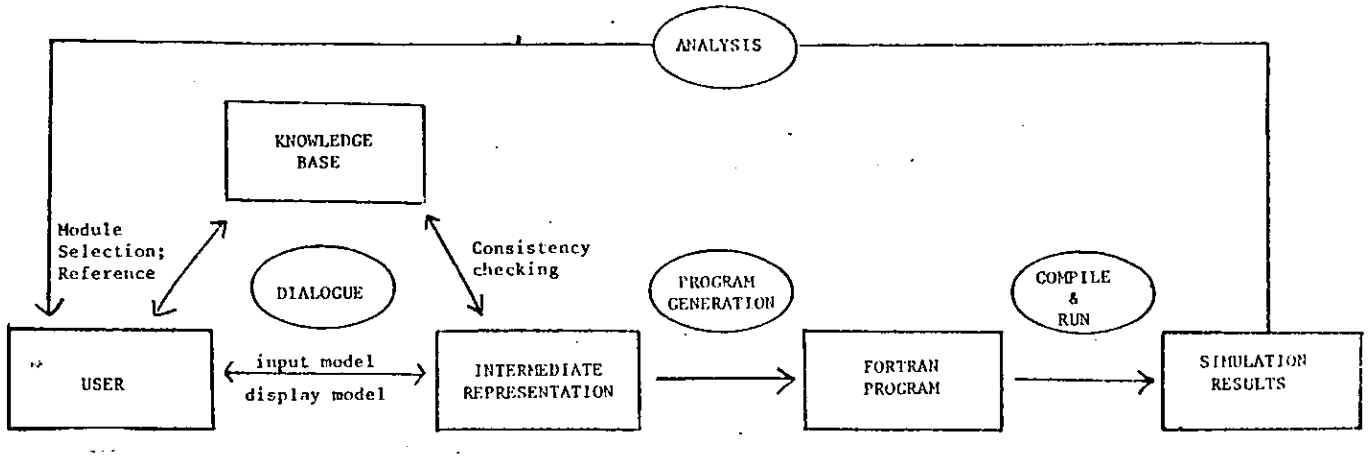

Figure 2-1: Structural Overview

Schematically, a System Dynamics model is represented as a directed graph with the compartments as nodes, and flows as arcs. Each compartment has an associated *state variable* whose value represents the contents of the compartment. Running a model consists of calculating the changes in the amount of material in each compartment, given some set of initial conditions and mathematical relationships governing the flows. The state variables are thus of central importance. Each compartment is labelled with a unique integer. Compartment number 0 is the source/sink. The contents of the *i*'th compartment are represented by the *state variable*: *Xi*. *Fij* represents a flow from compartment *i* to compartment *j* and is graphically represented by a directed arc from node *i* to node *j*. Each flow is governed by some well defined mathematical relationship which we shall call the *Flow Expression*.

Consider the following example (Fig. 3-1): we wish to model sheep grazing in a particular area. We represent the sheep and grass in terms of their calorific (energy) content. There are thus two compartments: *SHEEP* and *GRASS*. The material that flows is *ENERGY*. There are four significant processes represented in this model: photosynthesis, grazing, sheep respiration,* and grass respiration. Each is represented by a flow. The direction is indicated by the heavy arrows in the figure. Note that all of the flows except grazing involve the source/sink which is not explicitly represented in the figure.

The amount of material which flows at any given moment is determined by the Flow Expressions. In the tank analogy above, the Flow Expression is the regulator on the valve. The rate of flow at any time depends on any number of factors. The flow from one compartment to another often depends on the current contents of either or both of the donor and recipient compartments. For instance, the rate of energy production by photosynthesis is proportional to the amount of grass energy present. The coefficient of proportionality in this case is the specific rate of photosynthesis for grass. The user might specify a constant value for this rate, or s/he may want to incorporate more depth in the model by making the coefficient depend on other factors. The rate of photosynsthesis, for example, may be a function of such things as net radiation, and perhaps temperature as in the example. There may be other ways to compute the rate of photosynthesis. The user selects from a library of modules to specify how it is to be calculated. A module is a mathematical function with an associated ecological context describing

*"To respire" here is "to metabolize", *not* "to breathe"

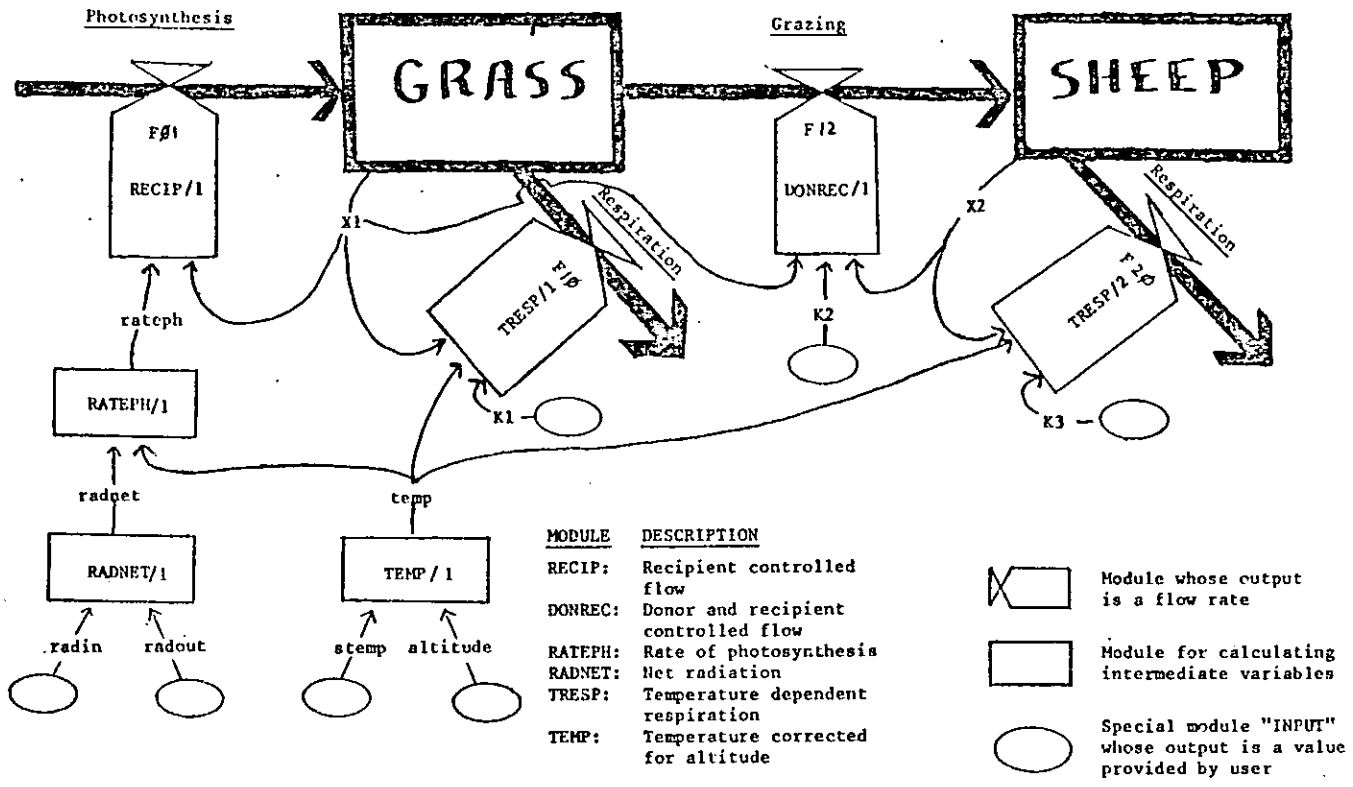

Figure 3-1: Example of a System Dynamics Model

the inputs and output(s) indicating how and when the module may be used in a model. Once a module is chosen, some means for obtaining values for each of its inputs must be specified. It may be a state variable, it may be assigned a constant value, or it may be computed using still another module. This extended dependency defines the mathematical substructure of a System Dynamics model.

4. The Intermediate Representation

The intermediate representation encodes the user's ecological description as a well-defined mathematical model. This representation is fairly close to the user's view of the problem, allowing for a more natural dialogue to take place and thus minimizing the amount of processing necessary while creating and modifying it. Small changes in the user's problem description cause small changes in the intermediate representation, even though they might imply considerable changes in the simulation program. The intermediate representation encodes enough information to be translatable into a FORTRAN program which produces mathematically correct simulation results for the desired system.

The example in figure 3-1 depicts the core of an intermediate representation for System Dynamics models. It consists of two conceptually distinct, but somewhat interrelated directed graphs. The superstructure for the System Dynamics model consists of compartments and flows. These correspond to the nodes and arcs, respectively, in the first directed graph. It is represented in the figure by the large boxes and heavy dark arrows. The substructure, which defines the mathematical relationships governing the flows, is an *acyclic* directed graph. We shall refer to it as the *flow graph* to distinguish it from the first graph. The nodes will be called *flow nodes*. Each flow node is associated with a module: a directed arc in the flow graph specifies how an input to a module is to be obtained. For example, the arc labelled "radnet" in figure 3-1 indicates that module "RADNET" is used to

compute an input to the module "RATEPH". There are four types of node in the flow graph:

- *Intermediate Node*: Has incoming arcs, one for each input to the associated module. Has at least one outgoing arc which provides an input to another node. These appear in the figure as small rectangular boxes.
- *Root Node*: Same as above except that the outputs from these nodes are used to calculate the amount of material which flows from one compartment into another at any given time. They appear in the figure schematically as valves resting on the arcs in the *other* directed graph. No arcs in the flow graph start at a root node.
- *Leaf Nodes*: These are associated with special modules which have no inputs. Thus, leaf nodes have no incoming arcs. There are two types:
  * *State Variable Nodes*: The output from this type of node is the current value of a state variable. They appear in the figure as large boldface rectangular boxes. They coincide with the nodes in the first directed graph. (eg. X1 from the GRASS compartment)
  * *Input Node*: The output from this type of node is a value provided by the user. They are fixed parameters in the model. They are small ovals in the figure.

A particular module can be used in a variety of contexts. The inputs used each time may or may not refer to the same quantity in the model, and correspondingly to the same variable in the eventual FORTRAN program. Similarly, different modules may share inputs. This can be seen in the example model: the module computing the rate of photosyntheses and the two uses of the module *Tresp* all use the same temperature as one of their inputs. On the other hand, the proportionality coefficient input *K* to module *Tresp* is different for both uses. This is reasonable, since in one case, we have plant respiration, and in the other, animal respiration.

The flow nodes can be thought of as *instantiated* modules. That is to say, a particular use of a module in a particular situation. To distinguish two nodes which use the same module, a node is named by placing a number at the end of the module name and incrementing the counter for each new use. This can be seen in figure 3-1. If all the inputs for more than one use of a module are the same, then it is the same instantiation, and hence one node with more than one outgoing arc (one for each use). The *Temp/1* node in the figure is such a node.

The requirements for an intermediate representation as described above are satisfied by a complete specification of these two interrelated directed graphs. Together they define the three major components of a System Dynamics model: the *Compartments*, the *Flows*, and the *Flow Expressions*.

5. Ecological Knowledge Base

The base of ecological data and relationships provides the user with the building blocks for making models and with guidance on how to put them together appropriately. The user consults this knowledge base directly when selecting modules. It is also used to infer the "obvious", easing the burden on the ecologist of specifying every last detail. For example, modules selected in certain situations

have some of the inputs automatically filled in on the basis of what has been previously specified in the model. For instance, one commonly used module in System Dynamics models is *Recipient controlled flow*, which indicates that the flow is proportional to the contents of the recipient compartment. Consider photosynthesis in figure 3-1. If photosynthesis has been entered in the model as a flow from the *source_sink* to the *grass* compartment and this module is selected, the state variable input (*GRASS* in this case) is automatically filled in, since the program *knows* what the recipient compartment is. The knowledge base is also used to ensure consistency in the model. For instance, the units of the quantities in the model must be compatible and conversions made when necessary. Modules selected must be applicable to the ecological context in which they are chosen.

The knowledge base is divided into three sections which are described below:

5.1. Module Library

These are mathematical functions, each with an associated ecological context for its use. They may be as simple as a one-line linear relationship, or arbitrarily complex. The main components of each module are:

- *Name*: Keyword to refer to module by.
- *Text*: Description of the ecological context for which this module is appropriate. Possibly details of its mathematical form.
- *Inputs*: Name and description of each input
- *Outputs*: Name and description of each output
- *Extra Information* (if available): default parameter values, reference to literature of studies supporting the use of the module and parameter values, author of method, species involved in the study.

There are currently about forty modules in the library ranging over a modest variety of applications.

Modules from example model (Fig. 3-1):

- *Name*: Tresp
- *Text*: Temperature dependent respiration; *Resp = K * Temp * Xi*
- *Inputs*:

Temp	-Temperature	-degrees C	-global
K	-Coeff of proportionality	-none	-local
Xi	-Energy of lifeform	-Calories	-local

- *Outputs*: Resp - Respiration of X

- *Name*: Temp
- *Text*: Temperature corrected for altitude; *Temp=Stemp-(3*Alt/1000)*
- *Inputs*:

Stemp	-Sea level temperature	-degrees C	-global
Alt	-Altitude above sea level	-meters	-local

- *Outputs*: Temp

The input descriptions are worth explaining in greater detail. For each input, there is the *conceptual name*, a verbal description, units (if any), and whether or not the input is likely to be a local or global variable. Each time a module is chosen and incorporated into the model, a *source* for each input must be specified. That is, some means for obtaining its value must be indicated. Consider the module *Tresp* above. The input *Temp* is likely to be the same one throughout the

model and is assumed to be so unless otherwise specified. Its source need only be provided once, no matter how many times the module is used: it is a global variable. On the other hand, the input *Xi* would usually represent a different lifeform for each use in the model. Thus, the same conceptual input can represent different quantities in the model. Each time the module is used, a source will have to be provided for that input: it is a local variable.

5.2. Entities

This includes all objects which the model can contain. For instance, lakes, sheep, soil, trees etc. At this point the entities section is very small, including only a few animals and plants. Eventually many more objects, including large taxonomies of plants and animals, will be in the knowledge base.

5.3. Process Library

This section contains all the processes which the user may incorporate into the model. This includes such things as grazing, respiration, evaporation, etc. Each process has rules determining what sort of entities can participate in that process, and in which direction(s) the flow may go. For instance, grazing takes place between animals and plants only, and energy flows from the plant entity to the animal entity. Again, the current process library is very small, but is ready for scaling up.

6. The Dialogue Subsystem

The model-building process begins with broad statements which indicate the main features of the model. The mathematical structure is specified, and finally, all initial conditions and parameter values are filled in. The user has access to the ecological knowledge base which is used to assist in selecting appropriate modules and parameter values. To some extent, we relieve the user of having to specify every last detail. Whenever possible, the program deduces the "obvious" quietly, behind the scenes (See beginning of section 5). At any point, the user can display the current state of the model in an easily readable format. The user's actions are continually monitored. Should any blatant inconsistency arise at any point, a message is printed which usually suggests remedial action. When the model is believed to be complete and ready for testing, the program invokes the consistency/completeness checker. The user is informed of any inadequacies and given the opportunity to fix or complete the model.

In short, the job of the dialogue handler is to create the intermediate representation. This consists of creating the two directed graphs discussed in section 4. It involves specifying the *Compartments*, the *Flows*, and the *Flow Expressions*. Commands are needed which allow arcs and nodes in these directed graphs to be created and modified in a natural way. The following six simple command statements allow the user to create fairly complex models rather quickly.

- DOES: Specifies an action or process taking place between one or more entities.

- USES: Specifies how a quantity is to be computed.

- SET: Assigns a value to a quantity.

- UNIFY: Two quantities in the model which had previously been assumed to be different are unified.

- SPLIT: One quantity which was used in more than one place in the model is split into two or more different quantities.

- DISPLAY: Displays the current state of the model, or a specified part of it.

The first statement must be a *DOES* statement. This results in the specification of one or two compartments and one flow. Consider this example: *SHEEP GRAZE GRASS*. If the middle word of a three word statement matches a process in the process library, then it is a *DOES* statement. The first and third words are found in the entities section of the knowledge base and become compartments in the model. The entities must match the process, however. For instance, grazing takes place between animals and plants only. The direction of flow is deduced. Energy flows from the plant to the animal. The system responds to a *DOES* statement with a unique tag for the flow. It is used when later referring to that flow. The names need to be both meaningful and short to avoid taxing the user's memory and excessive typing.

After at least one *DOES* statement has been specified, the user has a number of options. S/he may: 1) specify another *DOES* statement, creating another flow and perhaps additional compartment(s); 2) *SET* the initial contents of a compartment; or 3) specify the Flow Expression for the flow via a *USES* statement.

A *USES* statement results in a number of changes to the intermediate representation. Firstly a node is created if it is not already in use. Next, one arc for each of the module's inputs must be created. If the variables are local, unique names are created for each (based on the conceptual name). If an input is global, the conceptual name is used as it is. (A unique name is created anyway and stored: it can be used if this variable is ever *SPLIT*, as described below.) These names appear in the model display and must be used when subsequently specifying their *Source* via a *USES* or *SET* command. When a new arc is created for a global variable which is already around in the model, any characteristics associated with it must be inherited in the new use. Generally, when these new arcs pop up in the display, their *Source* slots are unfilled and appear as "?". If the quantity whose *Source* is being specified already had a different *Source*, the user is asked to confirm the change. If confirmed, this may necessitate further changes in the intermediate representation. For instance, if a quantity was originally computed using another module, and the corresponding node is not used elsewhere, then the entire subgraph hanging off this unused node needs to be marked. The consistancy checker informs the user at the end of any unused nodes. The user then decides whether they should be deleted. S/he may mean to include them in the model.

The effect of the *SET* statement is similar to that of *USES*, except that instead of specifying a new node in the graph and adding more arcs, it causes a leaf node of a special type to be added to the intermediate representation: it sets the value of a node input to a fixed value (See section 4). If this represents a change, the ripple effects must be handled in the same way as for *USES*.

The assumptions about the inputs being global or local may not always be correct. The user has the ability to *UNIFY* two inputs which had previously been different. S/he has the option to specify a unique name for the quantity represented by the input or to accept a name suggested by the program. The old names are kept around in case the user has a change of mind. Conversely, an input to two different nodes which had previously been assumed to be global, and thus the same quantity with the same name may in fact be meant to be different. This command is the opposite of *UNIFY* and is called *SPLIT*. This presents a special problem: if two uses have the same name, how can the user distinguish between the two? This is solved by always keeping around a unique name for every new use of any variable. If they are ever split up, the unique names are there to allow the user to distinguish between different uses.

These commands provide a great degree of power and flexibility. The modeller can build complex models quickly and conveniently. A facility is also provided for saving and retrieving partially or completely specified models for incremental model development.

7. Program Generation

The task of generating correct FORTRAN code to implement the model is greatly simplified because there is a reasonably well-defined program template which can be used to run System Dynamics models simulations. The following is such a template:

I) Read parameter values and initial contents of Compartments
II) Main Loop: For each time increment:
  - Calculate flows
  - Update contents of compartments
III) Examine simulation results

This simple template will not be appropriate for ecological models beyond the framework of System Dynamicss.

The hard part is in calculating the flows. Each module is already precoded as a SUBROUTINE. Its arguments are the inputs to the module, followed by the outputs. For example, consider module *Tresp*. The corresponding SUBROUTINE statement is: SUBROUTINE TRESP(TEMP,K,XI,RESP). The program generator must be able to output the correct sequence of subroutine calls so that a variables used as inputs always have values already assigned. It also has to get all the input and output variables in the appropriate argument slots. For our example in figure 3-1 the correct FORTRAN program segment would be:

```
Node:
Radnet/1        CALL RADNET(RADIN,RADOUT,RADNET)
Temp/1          CALL TEMP(ALTITUDE,STEMP,TEMP)
Rateph/1        CALL RATEPH(RADNET,TEMP,RATEPH)
RecipC/1        CALL RECIPC(RATEPH, X1, F01)
Tresp/1         CALL TRESP(TEMP, KP, X1, F10)
DonRec/1        CALL DONREC(X1,X2,CONST, F12)
Tresp/2         CALL TRESP(TEMP, KA, X2, F20)
```

The statement sequence is obtained by topologically sorting the nodes. The consistency checker in the dialogue handler ensures that the directed graph is

acyclic. The correct order of the arguments is found by using the names of the inputs as specified in the module library. We call these names the *conceptual names*, and the names used in the program are the *instantiated names*, the latter being what the user sees when building the model. *Both* names must be included in the intermediate representation. The conceptual names must match with the names in the module specification, and the instantiated names are output in the FORTRAN code. As mentioned previously, there are two leaf cases. First, there are fixed parameters. These are constants which are read in at the beginning of the program rather than appearing as constants in the code. This gives the user the option to change them without having to regenerate and recompile the FORTRAN program. The second type of leaf case is the state variable. The name of the compartment is used to find the appropriate variable name. Recall the state variable *Xi* is the contents of compartment *i* (See section 3). These variables are created in the early stages of program generation when the compartments are processed.

Updating the contents of the compartments is performed by adding all the incoming flows and subtracting all the outgoing flows for each compartment for each time step.

8. Current Major Limitations/Future Work

- The *Process* and *Entities* sections in the knowledge base are toyfully small. Scaling these up is straightforward.

- We will make it easier for the user to select appropriate modules. For example, a statement such as *"temperature depends on altitude"* will cause ECO to find the module(s) or chain(s) of modules for calculating this relationship.

- We need to go beyond System Dynamics models. A more general, lower level modelling approach will have to be adopted: an appropriate corresponding intermediate representation is already being developed. It is imperative that the user interface remains at a high level, relatively close to the user's view of the model. Otherwise, the ecologist will be unable to create models conveniently if at all. Serious limitations of System Dynamics models include:
  - No facility for handling substructure exists. For instance, a deer population may be divided into several herds, each of which may be subdivided into age and/or sex classes.
  - No facility exists for doing calculations on varied time scales in the same model. Certain things ought to be calculated daily, others monthly, or yearly.

- The construction of ecological models takes place within a certain context. For example, a user may be building a forestry model, or population dynamics model. Future versions of ECO will attempt to deduce the appropriate context as the model is being specified. This information can then be used to assist the user by suggesting model specifications

appropriate to the current context.

9. CONCLUSION

We have described ECO, the prototype of an intelligent front end for ecological modelling. ECO helps a user build a customised, FORTRAN program which simulates the ecological processes in an ecosystem. The user interacts with ECO in a free-form dialogue about the ecosystem using only ecological terminology. The prototype system can only build System Dynamics models, but we intend to lift this restriction in future versions of ECO.

The main issues we have addressed in ECO are:

- The design of an intermediate representation capable of representing the description of a wide range of ecological problems.

- The design of a knowledge base which can hold established ecological modelling knowledge, in a form accessible to a mathematically naive ecologist.

- The design of a dialogue handler which can provide a user-friendly interface to the intermediate representation and the knowledge base.

- The design of a program generator which can convert the intermediate representation into a FORTRAN simulation program.

Because of the wide terminological gap between the ecologist user and the ecological model, it was vital to have an intermediate representation which could describe the user's problem and then be translated into the simulation program. The design of this intermediate representation in the prototype ECO was simplified by the existence of the System Dynamics notation for describing ecological models. This notation is readily understood by ecologists and covers a wide range of models. A formalization of this notation provided the intermediate representation for ECO.

Extending ECO beyond System Dynamics models will require a significant effort in enhancing and/or partially redesigning the intermediate representation. This work is already under way and showing promise. The architecture we have developed for ECO is likely to be applicable in modelling *non*-ecological systems as well. Systems which involve the linking together of physical components such as electronic, or mechanical systems would be particularly suitable.

The classic expert-system, production-rule architecture was not suitable for controlling the free-form dialogue or for generating the FORTRAN program, and we have had to explore new techniques for these tasks. In future, production rules may find application as inference rules for preparing the intermediate representation for the program generator.*

---

*Such inference was found useful in MECHO, [Bundy et al 79].

References

[Barstow et al 82]
D. Barstow, R. Duffy, S. Smoliar, S. Vestal.
An Overview of PHINIX.
In *National Conference on Artificial Intelligence*. AAAI, Pittsburgh, Pennsylvania, August, 1982.

[Bennet and Englemore 79]
Bennet, James S. and Englemore, Robert.
SACON: A Knowledge-Based Consultant for Structural Analysis.
In *Proceedings of the sixth IJCAI*, pages 47-49. International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.

[Bundy et al 79] Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.
Solving Mechanics Problems Using Meta-Level Inference.
Research Paper 112, Dept. of Artificial Intelligence, Edinburgh, 1979.
In proceedings of IJCAI-6 and Expert Systems in the Micro-electronic age.

[Forrester 61] J. W. Forrester.
Industrial Dymanics.
MIT Press, 1961.

[Weiss et al 82] Weiss S., Kulikowski C., Apte C., Uschold M., Patchett J., Brigham R., and Spitzer B.
Building Expert Systems for Controlling Complex Programs.
In *Proceedings of AAAI-82*, pages 322-326. American Association for Artificial Intelligence, 1982.